**CS2102 - Database Systems**

**Project (Part 1): Adazal, an online shopping website**

**CS2102 Team 56**

| Guo Ai | | A0177850W |
| --- | --- | --- |
| Leeroy Liu | | A0218491B |
| Qu Mingsi | | A0185084Y |
| Zhao Pengfei | | A0164637X |

**Table of Contents**

[**Section 1: The Entity-Relationship Data Model**](#_a8vq62l0z9ii) **2**

[Section 1.1: The Entity-Relationship Data Model (Simplified)](#_m1pdb9ok9ibp) 2

[Section 1.2: The Entity-Relationship Data Model (Full)](#_lfbi03gozeyc) 3

[**Section 2: The Relational Database Schema**](#_dbycvkt6j24y) **6**

[**Section 3: Justifications for non-trivial design decisions**](#_z6kjkmolyc18) **10**

[**Section 4: Uncaptured Constraints**](#_81hhg9pescrs) **12**

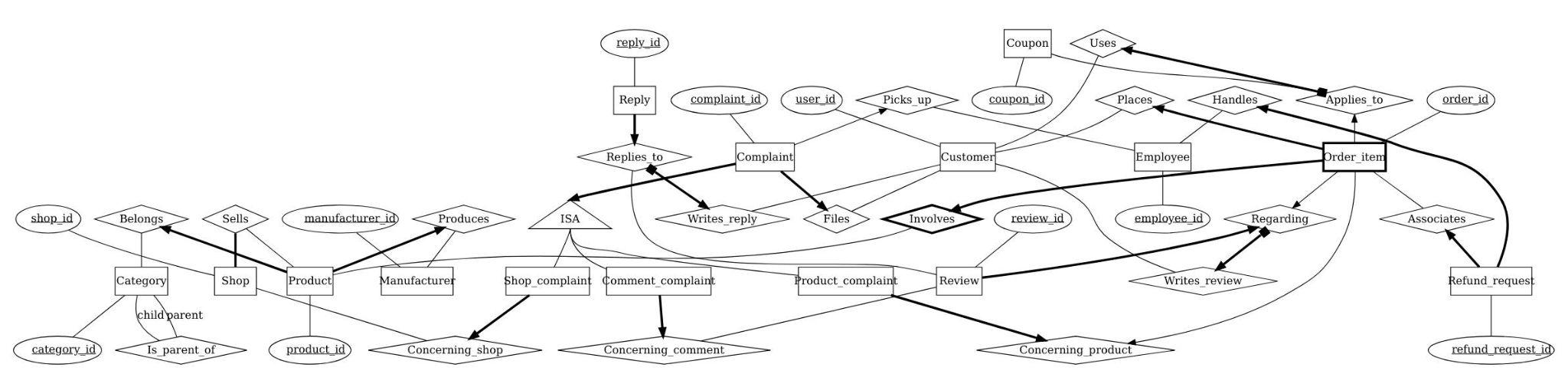
[Section 4.1: Constraints Not Captured by ER Data Model](#_lv2r8yq4qmr4) 12

[Section 4.2: Constraints Not Captured by Relational Database Schema](#_jqmr4vfg9dxr) 13

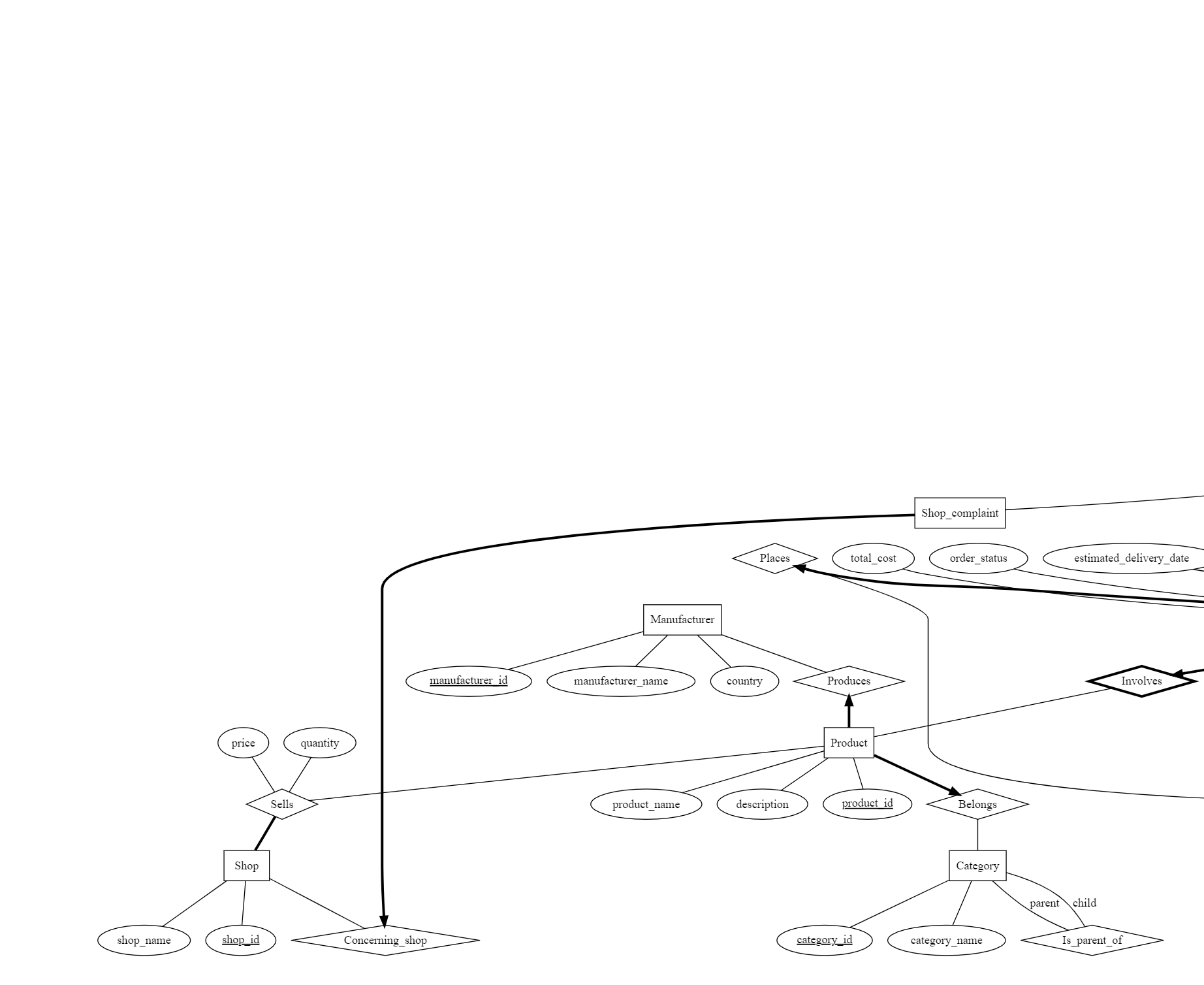
[**Section 5: All Relationships in the Entity-Relationship Data Model**](#_5794erte2517) **14**

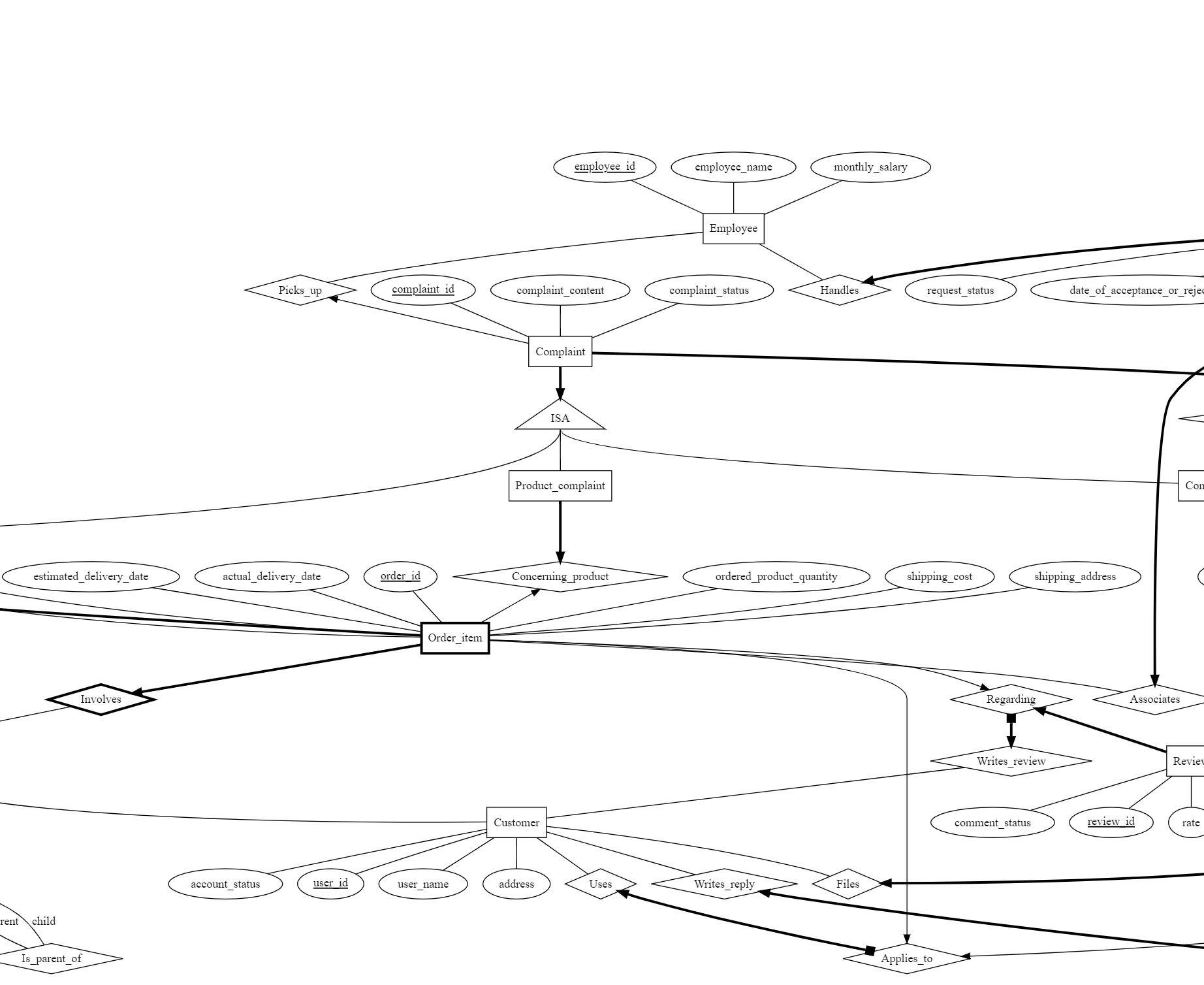
# **Section 1: The Entity-Relationship Data Model**

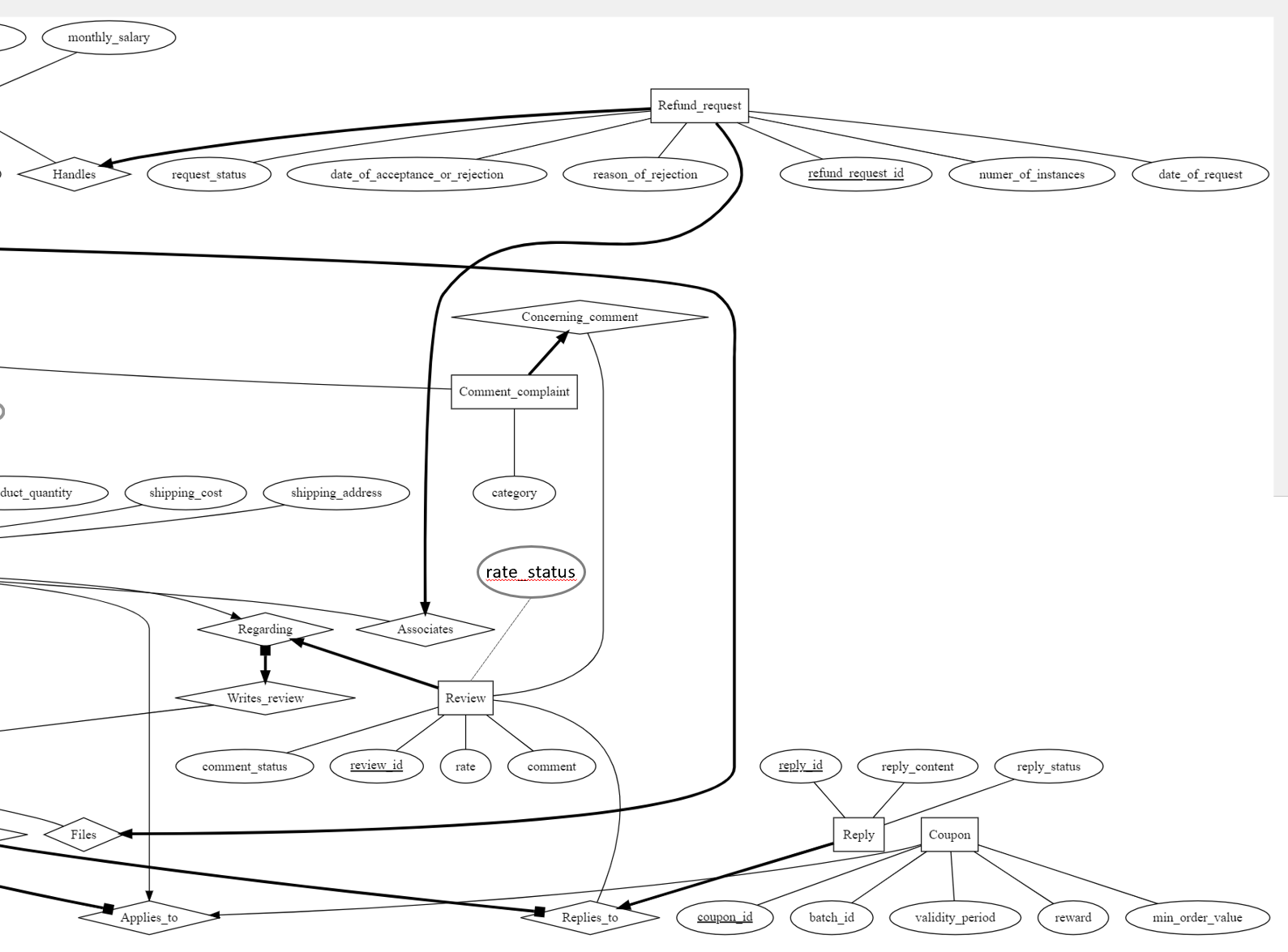
## **Section 1.1: The Entity-Relationship Data Model (Simplified)**



## **Section 1.2: The Entity-Relationship Data Model (Full)**







# **Section 2: The Relational Database Schema**

-- -- CS2102 Project (Part 1)

-- -- Group 56

-- -- For convenience. https://stackoverflow.com/questions/3327312/how-can-i-drop-all-the-tables-in-a-postgresql-database

-- -- Note that all names of all schema are public. This will delete all functions views, etc. defined in the public schema.

-- -- But it will not remove the system tables as they are in a different schema.

-- drop schema public cascade;

-- create schema public;

create table Shop (

shop\_id integer,

shop\_name varchar(50) not null,

primary key (shop\_id)

);

create table Category (

category\_id integer,

category\_name varchar(50) not null,

parent\_category\_id integer,

primary key (category\_id),

foreign key (parent\_category\_id) references Category

);

create table Manufacturer (

manufacturer\_id integer,

manufacturer\_name varchar(50) not null,

manufacturer\_country varchar(50) not null,

primary key (manufacturer\_id)

);

create table Product (

product\_id integer,

product\_name varchar(50) not null,

category\_id integer not null, --key & total (Belongs)

manufacturer\_id integer not null, --key & total (Produces)

product\_description text,

primary key (product\_id),

foreign key (category\_id) references Category,

foreign key (manufacturer\_id) references Manufacturer

);

create table Sells (

shop\_id integer references Shop on delete cascade,

product\_id integer,

quantity integer not null,

price float not null,

primary key (shop\_id, product\_id),

foreign key (product\_id) references Product on delete cascade

);

create table Employee (

employee\_id integer,

employee\_name varchar(50) not null,

monthly\_salary float not null,

primary key (employee\_id)

);

create table Customer (

user\_id integer,

user\_name varchar(50) not null,

account\_status varchar(20) default 'active' not null,

user\_address text,

primary key (user\_id),

check(account\_status in ('active', 'deleted'))

);

create table Order\_item ( -- this is an order table for order items

-- if a product in the product table is deleted,

-- then the order item will also be deleted (weak entity)

order\_id integer,

user\_id integer not null, --key & total (Places)

product\_id integer, -- key & total (Involves)

ordered\_product\_quantity integer not null,

shipping\_cost float not null,

shipping\_address text not null,

total\_cost float not null,

order\_status varchar(20) default 'being processed' not null,

estimated\_delivery\_date date,

actual\_delivery\_date date,

primary key (order\_id, product\_id), -- weak entity

foreign key (user\_id) references Customer,

foreign key (product\_id) references Product on delete cascade, -- weak entity

check(order\_status in ('being processed', 'shipped', 'delivered', 'requested for refund')),

check((order\_status <> 'request for refund') or (ordered\_product\_quantity > 1))

);

create table Refund\_request (

refund\_request\_id integer,

order\_id integer not null,

product\_id integer not null,

employee\_id integer,

reason\_of\_rejection text,

number\_of\_instances integer not null,

date\_of\_request date not null,

request\_status varchar(20) default 'pending' not null,

date\_of\_acceptance\_or\_rejection date,

primary key (refund\_request\_id),

foreign key (order\_id, product\_id) references Order\_item

on delete cascade,

foreign key (employee\_id) references Employee,

check(request\_status in ('accepted', 'rejected', 'pending')),

check(

((reason\_of\_rejection is null) or (request\_status = 'rejected'))

and ((reason\_of\_rejection is not null) or (request\_status <> 'rejected'))

),

check(

((date\_of\_acceptance\_or\_rejection is null) or (request\_status <> 'pending'))

and ((date\_of\_acceptance\_or\_rejection is not null) or (request\_status = 'pending'))

)

);

create table Review (

review\_id integer,

order\_id integer not null,

product\_id integer not null,

rate integer,

comment text,

rate\_status varchar(20) default 'active',

comment\_status varchar(20) default 'active',

primary key (review\_id),

unique (order\_id, product\_id),

foreign key (order\_id, product\_id) references Order\_item(order\_id, product\_id)

on delete cascade,

check(rate in (1, 2, 3, 4, 5)),

check(rate\_status in ('active', 'deleted')),

check(comment\_status in ('active', 'deleted')),

check((rate is not null and rate\_status is not null) or ((comment is not null) and (comment\_status is not null)))

);

create table Writes\_review (

user\_id integer not null,

review\_id integer,

order\_id integer not null,

product\_id integer not null,

primary key (review\_id),

unique (order\_id, product\_id),

foreign key (user\_id) references Customer,

foreign key (review\_id) references Review(review\_id) on delete cascade, -- deletion of review\_id technically should not happen, because if a review (comment) is inactive, we can just change the 'comment\_status' attribute without deleting the tuple in the 'Review' entity. We are putting 'on delete cascade' here just in case the database manager wants to make such changes.

foreign key (order\_id, product\_id) references Order\_item on delete cascade

);

create table Reply (

reply\_id integer,

reply\_content text not null,

review\_id integer not null,

reply\_status varchar(20) default 'active' not null,

primary key (reply\_id),

foreign key (review\_id) references Review(review\_id)

on delete cascade,

check (reply\_status in ('active', 'deleted'))

-- It is possible that the referenced review\_id is a review with only rate and no comment. This constraint is not captured in this relational schema but we think might be enforced by more advanced features later.

);

create table Writes\_reply (

user\_id integer not null,

review\_id integer not null,

reply\_id integer,

primary key (reply\_id),

foreign key (user\_id) references Customer,

foreign key (reply\_id) references Reply on delete cascade, -- deletion of reply\_id technically should not happen, because if a reply is inactive, we can just change the 'reply\_status' attribute without deleting the tuple in the Reply entity. We are putting 'on delete cascade' here just to be safe.

foreign key (review\_id) references Review(review\_id) on delete cascade -- Similarly, the deletion of a review\_id technically should not happen, because if a review (comment) is inactive, we can just change the 'comment\_status' attribute without deleting the tuple in the Review entity. We are putting 'on delete cascade' here just in case the database manager wants to make such changes.

-- It is possible that the referenced review\_id is a review with only rate and no comment. This constraint is not captured in this relational schema but we think might be enforced by more advanced features later.

);

create table Coupon (

coupon\_id integer,

batch\_id integer not null,

min\_order\_value float not null,

validity\_period date not null,

reward float not null,

order\_id integer,

product\_id integer,

primary key (coupon\_id),

unique (order\_id),

foreign key (order\_id, product\_id) references Order\_item (order\_id, product\_id) on delete set null

);

create table Uses (

user\_id integer not null,

coupon\_id integer,

order\_id integer not null,

product\_id integer not null,

primary key (coupon\_id),

unique (order\_id),

foreign key (user\_id) references Customer,

foreign key (coupon\_id) references Coupon on delete cascade,

foreign key (order\_id, product\_id) references Order\_item (order\_id, product\_id) on delete cascade

);

create table Complaint (

complaint\_id integer,

complaint\_content text not null,

complaint\_status varchar(20) default 'pending' not null,

employee\_id integer,

user\_id integer not null, -- files

primary key (complaint\_id),

foreign key (user\_id) references Customer,

foreign key (employee\_id) references Employee,

check (complaint\_status in ('pending', 'being processed', 'addressed')),

check ((employee\_id is null) or (complaint\_status <> 'pending')),

check ((employee\_id is not null) or (complaint\_status <> 'being processed'))

);

create table ProductComplaint (

complaint\_id integer primary key references Complaint

on delete cascade,

order\_id integer not null,

product\_id integer not null,

foreign key (order\_id, product\_id) references Order\_item on delete cascade

);

create table ShopComplaint (

complaint\_id integer primary key references Complaint

on delete cascade,

shop\_id integer not null,

foreign key (shop\_id) references Shop on delete cascade

);

create table CommentComplaint (

complaint\_id integer primary key references Complaint

on delete cascade,

review\_id integer not null,

category varchar(50),

foreign key (review\_id) references Review(review\_id) on delete cascade

);

# **Section 3: Justifications for non-trivial design decisions**

There are various heuristics involved in the design process to render the design suitable for our application. Listed below are some of the most important design decisions made.

**First,** we have used the concept of lazy deletion where appropriate, where a supposedly deleted instance only has a flag flipped to indicate its deleted status without actually deleting this instance, so that the relevant information associated with a ‘deleted’ instance will remain unaffected. For instance, a user may choose to close his/her Adazal account, and after the account closes, the comments, ratings and replies made by the user, according to the requirement of the application, shall remain visible to other users, just that they are shown as coming from ‘A Deleted User’. As such, our lazy deletion design, in this case, involves adding an account\_status attribute to the Customer (representing User, not using User since it is a reserved keyword in PostgreSQL) entity set. When the user account closes, the account\_status is simply changed from ‘active’ to ‘deleted’, thereby preserving all the relevant attributes, associated relationship sets, etc. When the system displays the user name, the system may simply check if the account\_status is ‘active’ or ‘deleted’. If the account\_status is ‘active’, the system will simply display the user name where needed; if the account\_status is ‘deleted’, the system will then simply display the user name as ‘A Deleted User’ where needed.

Another example would be the deleting of ratings, comments and/or replies, where the revision history needs to be preserved for auditing purposes. In this case, we have rate\_status and comment\_status attributes in the Review entity set, as well as reply\_status in the Reply entity set, that takes the value of ‘active’ when they are not deleted, and takes the value of ‘deleted’ when they are deleted. As such, this avoids removing the instances of the Review entity set and/or the Reply entity set, and it allows the actual content in the instances of Review entity set and/or Reply entity set to be preserved for auditing purposes.

**Second,** we have made several relationship sets into aggregations. From tutorial 3 on ER model question 1, we learned that there are 3 approaches to model a 3-way relationship among 3 entities.

Design A is to model the relationship set as a ternary relationship associated with the 3 entities. However, this design is less general than aggregations and makes it more difficult to apply the key or total participation constraint. Therefore, we did not use the ternary relationship set.

Design B is to model the relationship as 3 binary relationships between each pair of entity sets. However, this does not capture accurate semantics. If we want to use the 3 binary relationship sets to capture the 3-way relationship that “User U uses coupon C for order O”, it will mean that “User U uses coupon C”, “Coupon C is applied to order O”, and “User U places order O”, but does not necessarily mean that “User U uses coupon C for order O”. Therefore, we do not use 3 binary relationship sets.

Design C is to model the relationship using aggregations. First, we use a binary relationship set to link 2 entities, and then use aggregation on this relationship, and treat it as a virtual entity to participate in another relationship set. We decided to use aggregations because all the relationship sets are binary, which makes it easy for us to apply the key and total participation constraints. This also captures the semantics among the 3 involved entities accurately.

In the above “User uses a coupon for an order” example, we first connect the “Coupon” and “Order\_item” entity using the “Applied\_to” relationship set. This is because coupons and orders are closely related as the order total value must reach the minimum cost required by the coupon. Then we treat the “Applied\_to” relationship set as an aggregation. Both “Applied\_to” aggregation and “User” entity set participate in the “Uses” relationship set.

Similar aggregations include: 1) “Review” entity and “Order\_item” entity are first grouped together by the “Regarding” relationship, which is an aggregation. Both “Regarding” virtual entity set and “User” entity participate in the “Writes\_review” relationship set; 2) “Reply” entity and “Review” entity are first connected by the “Replies\_to” relationship, which is an aggregation. Both “Replies\_to” virtual entity set and “User” entity participate in the “Writes reply” relationship set.

**Third,** we have used the IS-A relationship on the entity Complaint and its subclasses. Since there are three types of complaints and there are some common attributes (eg. complaint\_content, complaint\_status, employee\_id) among them, while each of them has special attributes (eg. order\_id and product\_id for ProductComplaint, shop\_id for ShopComplaint, review\_id and category for CommentComplaint) as well, the entity Complaint well fits Object-Oriented Structure. Hence we chose the IS-A relationship for Complaint. A complaint must be either a product complaint, a shop complaint, or a comment complaint, hence it satisfies the covering constraint. A complaint cannot be a product complaint, shop complaint, or comment complaint at the same time, hence it does not satisfy the overlap constraint. Therefore a notation of directed thick edge is used from Complain to ISA triangle.

**Fourth,** we have modelled the Order\_item entity as a weak entity, which depends on the owner entity Product. The Order\_item entity has a partial key order\_id, but it is not globally unique. Order\_id is only unique when the product\_id is given. An order can have multiple products, so many order items can share the same order\_id. We must combine the order\_id together with the primary key (i.e., product\_id) of its owner entity (i.e. Product) to uniquely identify each tuple in the Order\_item entity. The identifying relationship is Involves, which is a many-to-one relationship. One product can be in many (0 or more) order items, and an order item must have exactly one product. In addition, when a product is deleted, it also makes sense that the associated order item containing this product should also be deleted. Therefore, we have included “on delete cascade” in the Order\_item entity when referencing the product\_id in the Product entity set.

**Fifth,** we have used several types of relationship constraints. For example, 1) Total participation constraint (at least one): Each shop must sell at least one product, hence there is a total participation constraint between the Shop entity and Sells relationship. The Sells relationship between Shop and Product entity sets is an “at least one-to-many” relationship;

2) Key constraint (at most one): Each complaint can be picked by 0 employee (when the status is “pending”) or 1 employee (after the status is “being processed”), hence there is a key constraint between the Complaint entity and Picks\_up relationship. The Picks\_up relationship between the Employee entity and Complaint entity is a “many to at most one” relationship;

3) Key and total constraint: Each product must belong to exactly one category, hence there is a key and total participation constraint between the Product entity and Belongs relationship. The Belongs relationship between Product and Category entity sets is a “one to many” relationship.

More constraints can be found in the ER diagram. Key constraints are represented using arrows; total participation constraint is represented using bold lines; key and total participation constraints are represented using an arrow-and-bold line.

**Sixth,** to characterise the different roles played by different instances of the same entity set, there are two reasonable approaches presented in the lecture. One would be the managers of employees case in SQL-1 Lecture, where manager\_id is placed as an attribute in Employees entity set and we impose a foreign key constraint on the manager\_id to reference the Employees entity set itself; Another approach would be the senior and junior mentors case in ER Lecture, where there is an explicit relationship set Mentors with foreign key constraints imposed on the seniors and junior student\_id to reference the Students entity set. In this project, we have selected the former approach to handle roles played by different instances of the same entity set, which can be seen in the Category entity set, to lower the complexity of the Entity-Relationship data model due to one less relationship set involved. Furthermore, it will be more performant if we wish to trace down a particular chain of categories into child categories, or if we wish to trace up a chain of categories into the parent categories, as we will only need to reference the one Category table, without the need to reference another table to query the parent/child categories of the current concerned category.

**Seventh,** we have combined the rate and comment into one entity set, known as Review due to their similarities, while keeping reply as its own entity set, known as Reply. Rate and comment, as can be seen in the application requirement, are similar and have roughly the same constraints imposed. For instance, each user can only rate or comment once on the product purchased by the user; The ratings and comments can both be modified or deleted at any time. Hence, it is sensible to combine them into one single Entity Set. Reply, however, is rather different, in the sense that a user can reply multiple times, and the reply can be and can only be directed to comments. Hence, it is reasonable to design Reply as a separate Entity and makes it participate in the relationship set RepliesTo to review, with key and total participation constraints imposed such that each reply must reply to one and only one review (i.e. the comment, since a reply cannot be directed to the rating).

**Eighth,** For the Order\_item entity set, we have incorporated an additional attribute of actual\_delivery\_date, for the ease of implementation of the constraint “Within 30 days from the delivery date of a product, the user may return the product for a refund.” This is because we may compare the difference between the attribute value estimated\_delivery\_date and actual\_delivery\_date to evaluate whether it has exceeded the 30 days threshold.

# **Section 4: Uncaptured Constraints**

## **Section 4.1: Constraints Not Captured by ER Data Model**

**First,** we did not enforce the “number of” products that shop sells, which is interpreted as “each shop must sell at least 2 products”, but our ER diagram only indicates total participation constraint (for Shop with respect to Sells, i.e., a shop must sell at least 1 product, not the intended “at least 2”), and is not enforced in our relational database schema either.

**Second**, we did not enforce constraints relating to the comparison of attributes between more than one table. For example, it is said that “Within 30 days from the delivery date of a product, the user may return the product for a refund.” This is not captured because it would require a comparison between the date of request in the Refund\_request table and the actual delivery date in the Order\_item table. As these 2 attributes are in different tables, it is not possible to check constraints for the attribute.

Besides, another example would be that the constraint ‘a user can file a complaint if he/she has not received a product that has been shown to be “delivered” in an order is not captured since it requires a comparison between the order\_status in Order\_item table and the Complaint table.

Additionally, another example is that when applying coupons, we need to check whether the total cost of the order has reached the minimum order value required by the coupon. This cannot be captured because it needs to compare the “total\_cost” attribute in the Order\_items table and “min\_order\_value” in the Coupon table.

Furthermore, it is required that “total\_cost” in the Order\_item table + “reward” in the Coupon table = sum of “price” in the Sells table for all the products in the order. Since this involves the comparison of attributes in multiple tables, the constraint cannot be captured by both ER data model and relational schema.

**Third,** all the check constraints written in the relational schema are not captured. For example, the request status of the funding request in ('accepted', 'rejected', 'pending') are not recorded in the ER diagram.

**Fourth,** referencing the same table for different records cannot be reflected in the ER diagram, such as different coupons of the same batch should have the same validity period and reward, but cannot be enforced as check constraints cannot compare between records of the same table. This constraint is not enforced in both the Entity-Relationship data model and relational schema.

**Lastly,** the description of the problem is “each order can apply at most one coupon”. However, since we modelled the order\_item as an entity (with the primary key (order\_id, product\_id)), the total participation constraint that ER diagram is capturing is actually “each order item can apply at most one coupon”, which is looser than the intended constraint. However, this constraint is enforced in the relational schema by setting “unique (order\_id)” in the “Coupon” and “Uses” tables.

## **Section 4.2: Constraints Not Captured by Relational Database Schema**

**First,** The total participation constraint between entities and relationships is not captured by the relational schema. While it is possible to enforce both key and total participation constraints in the relational schema via the use of combining the concerned entity table and relationship table into one, it is difficult to enforce total participation alone without the usage of more advanced mechanisms like trigger or assertions. For example, the total participation constraint in the Sells relationship set is not captured in the relational schema, where each instance of the Shop entity set must participate in at least one instance of the Sells relationship set. In our relational schema, it is possible that a shop does not appear in the Sells table, which violates the total participation constraint.

**Second,** another constraint that is not captured by the relational schema is the possibility of a reply to a null comment. Since we designed the Replies\_to relationship set that involves the Reply entity and the Review entity, and the Review entity is representing both the rating (rate attribute) and the comment (comment attribute), it is possible that a Review instance has a non-null rate and a null comment, and this will still constitute a valid Review instance that can be replied to through the Replies\_to relationship set. This is less desired and the constraint that a reply should only be directed towards a comment, rather than a rating, is not captured by the relational schema. It is noteworthy that this constraint is not captured by the ER data model either.

**Lastly,** some constraints on the sequence of changes to attributes are not captured. For example, order\_status in the Order\_item table must be changed to “shipped” first before it can be changed to “delivered”. The value of order\_status cannot be “delivered” and then changed to “shipped”. Similarly, in the Refund\_request table, after the request\_status is changed to “accepted” or “rejected”, it cannot be changed to “pending”. Also, in the Complaint table, the sequence of “complaint\_status” must be first “pending”, then “being processed”, and lastly “addressed”. It is noteworthy that this constraint is not captured by the ER data model either.

# **Section 5: All Relationships in the Entity-Relationship Data Model**

In case the ER data model is a bit complex to read, we have listed all the relationships in our ER diagram. All our relationship sets are binary. In the middle is the relationship set, and on both ends, we have the 2 entity sets. The constraint on the left/right of the relationship set is the constraint between the left/right entity and the relationship. We have also indicated the is-a relationship, and which relationships are aggregations.

Shop - (total participation) Sells - Product

Product - (key and total participation) Belongs - Category

Manufacturer - Produces (key and total participation) - Product

Category (parent) - Is\_parent\_of - Category (child, itself)

Order\_item (weak) - (key and total participation) Involves - Product

Customer - Places (key and total participation) - Order\_item

Refund\_request - (key and total participation) Associates - Order\_item

Employee - Handles (key and total participation) - Refund\_request

Review - (key and total participation) Regarding (key) - Order\_item

Customer - Writes\_review (key and total participation) - Regarding (aggregation between Review and Order\_item )

Reply - (key and total participation) Replies\_to - Review

Customer - Writes\_reply (key and total participation) - Replies\_to (aggregation between Review and Reply)

Coupon - (key) Applied\_to (key) - Order\_item

Customer - Uses (key and total participation) - Applied\_to (aggregation between Coupon and Order\_item )

Customer - Files (key and total participation) - Complaint

Employee - Picks\_up (key) - Complaint

Complaint - ISA - Product\_complaint, Shop\_complaint, Comment\_complaint (satisfy cover constraint, do not satisfy overlap constraint, bold arrow)

Product\_complaint - (key and total participation) Concerning\_product (key) - Order\_item

Shop\_complaint - (key and total participation) Concerning\_shop - Shop

Comment\_complaint - (key and total participation) Concerning\_comment - Review